FinSearch

## A. Understanding the Data :

Functions used :

1. df.head()
2. df.shape
3. df.info()
4. df.describe()
5. df.value\_counts()

Operations Performed :

1. Changing the data type of column
   1. df [‘Column\_name’].astype(‘category’/float64/int64)

## B. Data Cleaning

Operations Performed :

1. The overall idea being cleaning the data, filling the anomalous values by NaN or appropriate values if possible.
2. Checking number of null values in individual columns
   1. null\_value\_column= df.isna().sum()
3. Checking duplicates
   1. df.duplicated().sum()

### Checking for unique values

* 1. df.nunique()

### Drop redundant columns

* 1. df.drop([‘Column Name’], axis=1, inplace=True)

### Handle Categorical variables

* 1. Selecting all categorical columns :
     1. cl = list(df.select\_dtypes('category').columns)
  2. Running for loop to find all random strings present in them
     1. for c in cl:
     2. print(c)
     3. print(df[c].value\_counts())
     4. print()
  3. Replacing the random string
     1. df['Column'] = df['Column'].apply(lambda x: np.NaN if x == "!@9#%8" else x)

1. Handel Numerical variables
   1. df[col] = df[col].apply(lambda x: x if x is np.nan or not isinstance(x, str) else x.replace("\_","")).replace("",np.nan)
2. Transform one column by using another column as bet
   1. df['Column'] = df.groupby('Column\_ID')['Column'].transform(lambda v: v.mean())
3. Can look at this function , isinstance(val, str) , it's useful sometimes
4. Can create a function to clean the data and apply it using df[‘column’].apply()

## C. Sanity Checks

1. The idea being checking individual columns using df[‘column’].describe() for very arbitrary values that doesn't make sense and cleaning them by using B. Cleaning Data

## D. EDA

1. The idea being analysing the data for outliers and clearing those data. For analysis we use boxplots
   1. plt.figure(figsize=(20,10))
   2. sns.boxplot(df[‘columns’])
   3. plt.xticks(rotation=30)
   4. plt.show()
   5. #df[‘columns’] -> here you can also pass array containing different columns to plot them together
2. The following code is used to remove outliers :
   1. def remove\_out(df\_clean, num\_cols, lbv=0.25, hbv=0.75):
   2. Q1 = df\_clean[num\_cols].quantile(lbv)
   3. Q3 = df\_clean[num\_cols].quantile(hbv)
   4. IQR = Q3-Q1
   5. lb = Q1-1.5\*IQR
   6. hb = Q3+1.5\*IQR
   7. for i in num\_cols:
   8. df\_clean = df\_clean[(df\_clean[i]>=lb[i]) & (df\_clean[i]<=hb[i])]
   9. return df\_clean
   10. —-----------------------------------------------------------------------------------------------
   11. df = remove\_out(df, cols, lbv=0.2, hbv=0.9)

## E. Data Processing

1. The idea is creating encodings for str objects
2. Replacing string value by integers
   1. df['Column'] = df['Column'].map({'Bad': 0, 'Standard': 1, 'Good': 2})
3. Creating encoding of strings
   1. dummy\_df = pd.get\_dummies(df[‘column’], drop\_first=True)
   2. df = pd.concat([df, dummy\_df], axis=1)
   3. df = df.drop(['column'], axis=1)
4. Data splitting
   1. XX = df.drop("Credit\_Score", axis=1)
   2. yy = df["Credit\_Score"]
   3. # Using SMOTE to handle class imbalance
   4. sm = SMOTE()
   5. X, y = sm.fit\_resample(XX, yy)
   6. X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, stratify=y, train\_size=0.8, random\_state=1)
5. Scaling
   1. Can apply different scaling methods to normalise data
   2. Note : use fit\_transform on X\_Train and only transform on X\_Test

## F. Model

1. Create a model using the algorithm you want
2. Use confusion matrix to analyse the data
   1. a = {'Poor': 0, 'Standard': 1, 'Good': 2}
   2. list(a.keys())
   3. cm = confusion\_matrix(y\_test, y\_pred\_test)
   4. disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=list(a.keys()))
   5. disp.plot()
3. Plotting most important features
   1. *# Function to plot top 20 feature\_importances*
   2. def plot\_feature\_importances(feature\_importances, cols):
   3. features = pd.DataFrame(feature\_importances, columns=['coef\_value']).set\_index(cols)
   4. features = features.sort\_values(by='coef\_value', ascending=False)
   5. top\_features = features
   6. *# features = features.head(20)*
   7. plt.figure(figsize=(10, 6))
   8. sns.barplot(x='coef\_value', y=features.index, data=features)
   9. plt.show()
   10. return top\_features
   11. —----------------------------------------------------------------
   12. top\_featues = plot\_feature\_importances(model.feature\_importances\_\*100, X\_train.columns)

## G.HyperParameter Tuning

folds = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

param\_grid = {

'max\_depth': [5, 8],

'n\_estimators': [200, 250, 300],

*# 'learning\_rate': [0.001, 0.1],*

}

estimator = XGBClassifier(learning\_rate=0.1,

min\_child\_weight=1,

gamma=0,

subsample=0.9,

class\_weight='balanced',

random\_state=42,

n\_jobs=-1,

tree\_method='gpu\_hist')

xgb\_search = RandomizedSearchCV(estimator,param\_distributions=param\_grid, scoring='roc\_auc', cv=folds, verbose=2,random\_state=42, return\_train\_score=True, n\_jobs=-1)

xgb\_search.fit(X\_train, y\_train)

xgb\_best = xgb\_search.best\_estimator\_

xgb\_best

xgb\_best\_model = xgb\_best.fit(X\_train, y\_train)

y\_pred\_test = xgb\_best\_model.predict(X\_test)

accuracy\_score(y\_test, y\_pred\_test)